

# The Linux CryptoAPI

## A User's Perspective

David Bryson

May 31, 2002

### Abstract

Today we are going to talk about using strong cryptography inside Linux. With currently available kernel patches discussed in this article you can encrypt your entire hard drive, network connections, and swap space. We will be discussing two of the solutions and focusing specifically on one: The Linux CryptoAPI.

Cryptography is the science of designing and analyzing ciphers. A cipher is an algorithm to transform *plaintext* into an unreadable mishmash, called *ciphertext* using a predetermined code sequence. The intent is that only the author and the intended recipient have access to the key and are thus the only ones with access to the plaintext. The process of applying the cipher to the text is called enciphering or encrypting, and the reverse process is deciphering.

To encrypt data using a given cipher a key is needed. The key provides the security of the cipher and only allows the holder of the key to decrypt whatever has been encrypted. Once you have a key it is fed into the cipher along with the plaintext to produce the ciphertext. Decryption is done in the same way except that the ciphertext is fed into the cipher along with the key and the plaintext is produced.

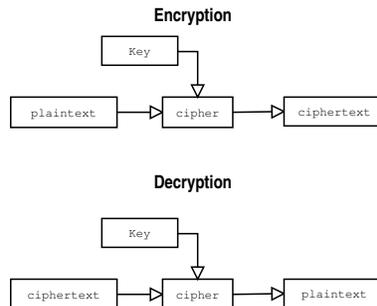


Figure 1: Encryption and decryption

Adding cryptographic support to kernel space gives you access to a lot of neat uses for information security. This article describes how cryptographic support may be added to the Linux kernel and how to secure your data using the functions of the CryptoAPI. Some of the uses for the CryptoAPI are:

- Encryption of all physical media, i.e. drive partitions, swap space, and CDROM images

Imagine that you are developing the next generation killer app for the Internet, and want to protect your work from prying eyes. Kernel level support for encryption of a hard disk partition allows you to encrypt the partition containing your home directories, so that the data stored on disk is unreadable, while still being able to access the data yourself.

- Encryption of network traffic, i.e. IPsec or any other form of end-to-end network encryption

Virtual Private Networks (VPNs) allow for network traffic to be encrypted before it leaves your computer. With VPNs becoming more widely used it is important to have kernel level support for the encryption in order to maintain acceptable network throughput and security.

## 1 Survey of cryptographic implementations

Currently there are two main sources of kernel level cryptography out there. I will briefly discuss loop-AES but focus on the CryptoAPI. The nice thing about both of these packages is that they are plugged into the kernel without the need for much modification to the actual sources(or rebooting!). The only modification that must be made directly to the kernel sources is a patch for the loopback driver which can be easily loaded/unloaded as a module. Hopefully this patch will be merged into the mainstream kernel soon.

Both packages install a modified loopback module for the kernel. This modified loopback module intercepts read/write requests to the encrypted filesystem and decrypts/encrypts the data as required. We discuss this in more detail in later sections.

Loop-AES is only designed for filesystem encryption and uses the Advanced Encryption Standard(AES) competition winner(the Rijndael algorithm) for its cipher.

It is quite fast and includes an implementation in assembly for Intel x86 processors(other architectures can simply use the C implementation). More information on loop-AES can be found at its SourceForge page, <http://loopaes.sourceforge.net>.

The CryptoAPI package provides a more general framework for kernel level encryption. It includes a generic interface to allow any other kernel module to encrypt data. File system encryption is but one of the applications, and CryptoAPI provides a sample implementation of this with the standard distributed

package. Along with its generic interface, CryptoAPI allows the user to select from 12 ciphers currently provided, as well as supporting the mounting, unmounting, and use of loop-AES encrypted filesystems.

## 2 Installation of CryptoAPI

Lets take a walk through getting it installed and setup on your machine. First we need to go and retrieve the latest CryptoAPI sources for your kernel. For this you'll need to have a copy of the sources for your kernel and know what version of the kernel that is. The latest CryptoAPI tarball can always be accessed at <http://www.kernel.org/pub/linux/kernel/crypto/>. Once you have retrieved the necessary versions go ahead and unpack the tarball file to `/usr/src/cryptoapi`:

```
$ cd /usr/src/cryptoapi
$ tar xvzf cryptoapi-0.1.0.tar.gz
```

**Note:** Always make sure to check out the README for any software you download!

Before we can compile any of the ciphers we need to patch our loop driver to support the encryption hooks. There are currently two loop patches distributed with the CryptoAPI. There is the loopiv patch, which includes minimal support needed for the CryptoAPI. And there is also the loop-jari patch which is distributed with loop-AES, it includes support needed for CryptoAPI plus some additional bug fixes. I will be taking the minimalist approach and only worry about one patch, loop-iv(Initialization Vector), which provides the support for what we need. To do this go to the top level directory of the CryptoAPI sources and use the makefile to do the following:

```
$ make patchkernel KDIR=<kernel source dir> LOOP=iv
```

**Note:** If you do not have the exact version of the kernel that the loopiv patch is for chances are you are probably Ok. Since the loop driver hasn't changed much since the move from 2.2 to 2.4 you shouldn't have a problem applying the patch anyway. There is a script that attempts to find the closest(lower) version patch to your kernel tree and it will patch it automagically. To patch with the loop-jari, enter 'jari' instead of 'iv' for the LOOP parameter.

Now we must first recompile our loop driver so that it is aware of the CryptoAPI. If you have a kernel configuration already setup then just recompile and install your loop modules. Otherwise you will have to compile the module by hand with some C compiler flags.

If you did everything right you should see a few lines go by saying that patches were applied successfully. Now we are ready to compile the cipher modules and install them into your system. We can compile the modules for all the ciphers and the cryptographic loopback plugin and install them without recompiling our kernel with the magic of kernel modules. To compile the modules you simply type:

```
$ make modules KDIR=<kernel source dir>
```

Then the CryptoAPI modules will begin to build. After which we need to install the modules in the appropriate kernel modules directory so that the kernel can load them. There is an autoinstall script which puts the modules in the appropriate directory, execute it with:

```
$ make install_modules
```

There is one more piece of software(non-kernel) that needs to be compiled before we can use the loopback encryption support. The `losetup` program, which is a part of the `util-linux` package, needs a patch to tell the kernel what cipher and key to use for your encrypted devices. There are different solutions for this depending on what Linux distribution you are using. If you are running Debian Woody then the `losetup` patch is already included in the `utillinux` package on your system. Debian Potato users can download the patch and recompile the `util-linux`. There are patched rpms available for Redhat systems as well as patches for the source tarball at <http://www.kerneli.org/cryptoapi>. At the time of this writing, Redhat 7.3 does not come with `losetup` crypto support, but we do provide the needed rpms(for both Redhat 7.2 and 7.3).

If your distro doesn't yet come with the patch included submit a patched version for the next revision. Once you have completed the `losetup` upgrade The CryptoAPI is now installed on your computer and you are ready to start setting up encrypted devices!

### 3 Using CryptoAPI

In this section we will talk about one application of CryptoAPI: encrypting a whole filesystem. On most installations the hard drive is split into several partitions. Each partition may then be mounted on an empty directory. One arrangement might be:

Partition	Mount point
/dev/hda1	/
/dev/hda2	swap space
/dev/hda5	/usr
/dev/hda6	/var
/dev/hda7	/home

Suppose you want to encrypt the partition containing your home directories, `/dev/hda7`. This means that every piece of information that gets written to that partition will be encrypted: plain text, binary files and even filesystem metadata (things like inode information, file and directory names and so on).

In the introduction we mentioned that CryptoAPI supports filesystem encryption using the loopback device; we shall now explain how that works. The traditional way to mount a filesystem, either at boot time, or later on, is with

the mount command. For example we can mount the home directories with the command:

```
$ mount -t ext2 /dev/hda7 /home
```

This tells the kernel that the device `/dev/hda7` contains an ext2 filesystem (the plain vanilla linux filesystem) and that all requests for files under `/home` should go to that partition. The loopback device allows for a level of indirection when mounting a filesystem. Instead of mounting the filesystem directly on `/home` one can mount the filesystem on the loopback device, and then mount the loopback device on `/home`. This has the effect of sending all kernel commands to the filesystem through the loopback device.

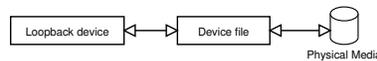


Figure 2: The loopback device

It is at this stage that the commands may be intercepted and altered as needed to perform encryption (for a write command) and decryption (for a read command). This is where the cryptoloop driver comes in. After everything had been mounted, every command to the filesystem is intercepted by the loopback device, sent to the cryptoloop device where it is processed, and then passed on to the filesystem. Now we can encrypt all the data on the drive!

You can also think of the way that it plugs in from the diagram of encryption/decryption from earlier. The loopback device takes the place of the cipher, the cryptoloop feeds it the key, and the plaintext and ciphertext represent the filesystem data going through the device.

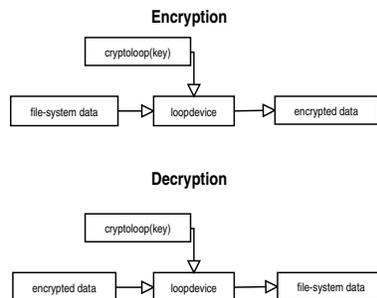


Figure 3: The cryptoloop filter

There are two different ways to encrypt files on your system. One, is using an actual device like an attached peice of storage(i.e. hard drive or CDROM image ). The other is to create a very large file, create a filesystem inside that file and

then mount the filesystem via the loopback device. We will be discussing the second method.

The first step is to create a large file to use for our filesystem; the larger the file, the more data we can store inside it, just like a regular hard disk partition. However, as for hard disk partitions, once we have created the file and made a filesystem inside, we cannot change the size without losing all the data.

In order to create a file we must fill it with data so that it takes up space on the hard disk. To use this we use random bytes generated from the `/dev/urandom` device. It is perfectly possible (and faster) to fill the file with zeros from `/dev/zero`. However this makes it easy for an attacker to see where the filesystem data is:

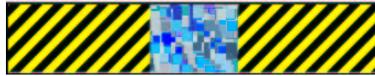


Figure 4: A file with zero's and random data

The striped graphics represent the zero's and the random squares represent the random looking data encrypted on your drive. As you can see this would give an attacker a considerable advantage over a storage device that looked completely random like this:

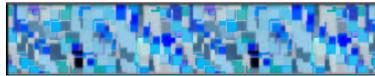


Figure 5: A file with all random data

Now that we have explained how it works, let's get everything setup. First we need to load the necessary kernel modules for the CryptoAPI into the kernel. This consists of four main modules:

1. `cryptoapi`, which provides the generic framework for different ciphers
2. `cryptoloop`, the interface for the ciphers and the loop driver
3. `loop`, the patched loop driver that the `cryptoloop` interfaces with
4. `cipher-x`, where `x` is the name of the cipher you wish to use

To load the first three modules we use the `modprobe` command:

```
$ modprobe cryptoloop
```

This should load the `cryptoapi`, `loop`, and `cryptoloop` modules into the kernel. You can check this using the `lsmod` command, and your output should look something like the following:

Module	Size	Used by	Not tainted
cryptoloop	1884	0	(unused)
loop	7664	0	[cryptoloop]
cryptoapi	3204	0	[cryptoloop]

Now we need to create the random data file using the `dd` command *before* we create a filesystem on the file. The kernel device `/dev/urandom` generates pseudo random data, using random data provided by the `/dev/random` device. Unlike `/dev/random` it provides a constant stream of bytes, but the less entropy that is provided by `/dev/random` the less random `/dev/urandom` will be. It works best if it has a stream of random events like mouse movement. So when executing the following command try to move your mouse or type a lot on the keyboard. Both of these contribute to the entropy pool.

```
$ dd if=/dev/urandom of=/home/mutex/cryptofile bs=1M count=50
```

This will create a 50 megabyte file named “cryptofile” in my home directory. Unfortunately this is not a fast procedure and will take good amount of time depending on the speed of your system. Next we need to load a cipher module for the encryption(in these examples I will be using the twofish cipher). All the module names for the ciphers are prepended with “cipher” so to load twofish we execute `modprobe` again:

```
$ modprobe cipher-twofish
```

Now we can mount the file like a regular device and for this we use the `losetup` command:

```
$ losetup -e twofish /dev/loop0 /home/mutex/cryptofile
```

The `-e` flag specifies the cipher used, in this case twofish; `/dev/loop0` is the loop device that we are using and `/home/mutex/cryptofile` is the filename. This will prompt you for a keysize to use and a password, which I entered '128' and then my password. In general, the larger the keysize(for a given cipher) the harder it is to decrypt your data using a brute force attack. However due to different enciphering algorithms a 128-bit key on one cipher may be even stronger than a 1024-bit key in another cipher.

Your password is used to generate the key, but it need not have the same number of bits as the key. It would be very tedious to type in a 128 character password in order to generate a 1024-bit key. Instead the password is used to generate a key via a hashing function.

```
Available keysizes (bits): 128 192 256
```

```
Keysize: 128
```

```
Password :
```

The password can be any length. Next we need to create a filesystem through our loopback device on the cryptofile with the `mkfs` command:

```
$ mkfs -t ext2 /dev/loop0
```

Naturally you can create any filesystem you desire, I just happened to choose ext2. Finally we can mount our encrypted filesystem just like a regular device using the mount command!

```
$ mount -t ext2 /dev/loop0 /mnt/crypto
```

You should be able to copy files and directories just like a normal mounted drive. Now you have a completely encrypted filesystem to play with! A common question on the Linux-crypto mailing list is, “How do I do an encrypted root partition?” While it *is* possible to do this with the cryptoloop device it is not recommended. The performance impact would be extremely bad and it requires a bit of working around initialization procedures when the system boots. It makes much more sense to only encrypt small peices of information which are sensitive: like a home directory for example.

Make sure that when you unmount your encrypted drive you unlink the loop device as well so that if somebody were to come along and try to mount it they would need the password.

```
$ umount /mnt/crypto
$ losetup -d /dev/loop0
```

It is easy to automate some of these steps with shell scripts so that you don't have to exectue two commands for mounting/unmounting.

## 4 CryptoAPI, past present and future

At the time of this writing the CryptoAPI is about to release its first tarball distribution as a branch from the original international kernel patch(kernelint). The original kernelint patch was designed to do just filesystem encryption and was written for the 2.2 series kernels. But after a while it evolved into a general purpose API for use in any kind of kernel space encryption. This movement caused the project to lie in hybernation for about 6 months before a new code-base was released. But with a new team of developers progress is starting to pick up again.

Some things to look forward to in the future from the CryptoAPI:

- Readability code changes: Some of the ciphers are taken from various places and not implemented by the CryptoAPI team. Thus they are inconsistant in the style of implementation and hard for a programmer to read. Rewrites to streamline the look of the code are in progress.
- Assembly implementations for ciphers on various architechtures: Adding assembly implementations of the ciphers will increase execution speed and provide a smaller memory foot print. This is especially useful in high-load situations where every process cycle counts.

- Cryptographic Hardware support: Several vendors manufacture cryptographic algorithms implemented in hardware. This usually results in performance that is 1000 times that of the fastest software implementations.

The new CryptoAPI codebase is still fairly young and needs some work. The project is always looking for fresh programmers with experience in cryptography or kernel programming. And those that can't program can always help by testing new patches and filing bug reports.

Hopefully this article taught you a bit about encryption and how the CryptoAPI can be used to secure your information. Look for releases and updates from the CryptoAPI project at our website <http://www.kerneli.org/>.